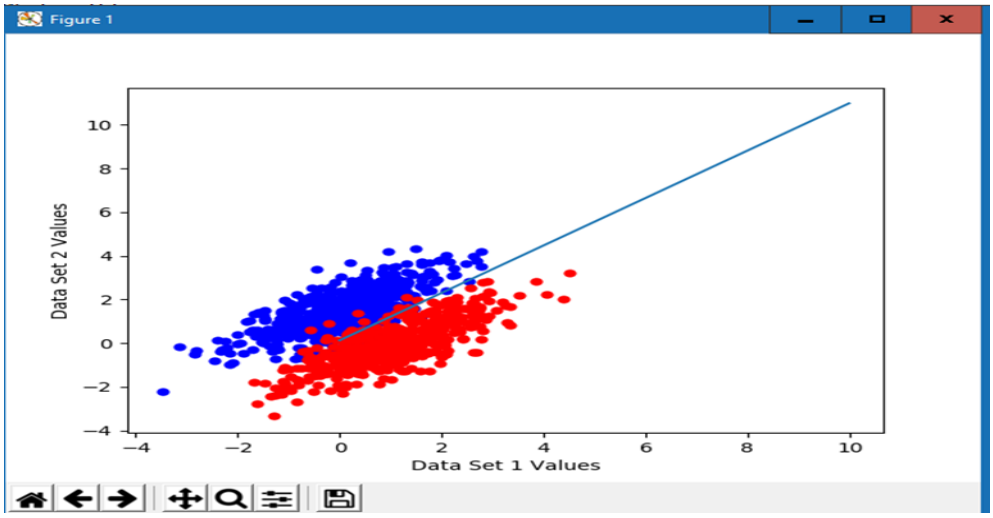
1. Perform batch training using gradient descent. Divide the derivative with the total number of training dataset as you go through iteration (it is very likely that you will get NaN if you don't do this.). Set your learning rate to be η= {1,0.1,0.01}. How many iterations did you go through the training dataset? What is the accuracy that you have? What are the edge weights that were learned?

Stopping the objective, the loop when the gradient is close to zero (the iteration reaches 10000) and η = 1 we have the following



The program went through 9,990 number of iteration and gave an accuracy of 0.8231 (82%) with edge weights learned: [-1314.62972534, -4224.39216911, 4584.21146125]

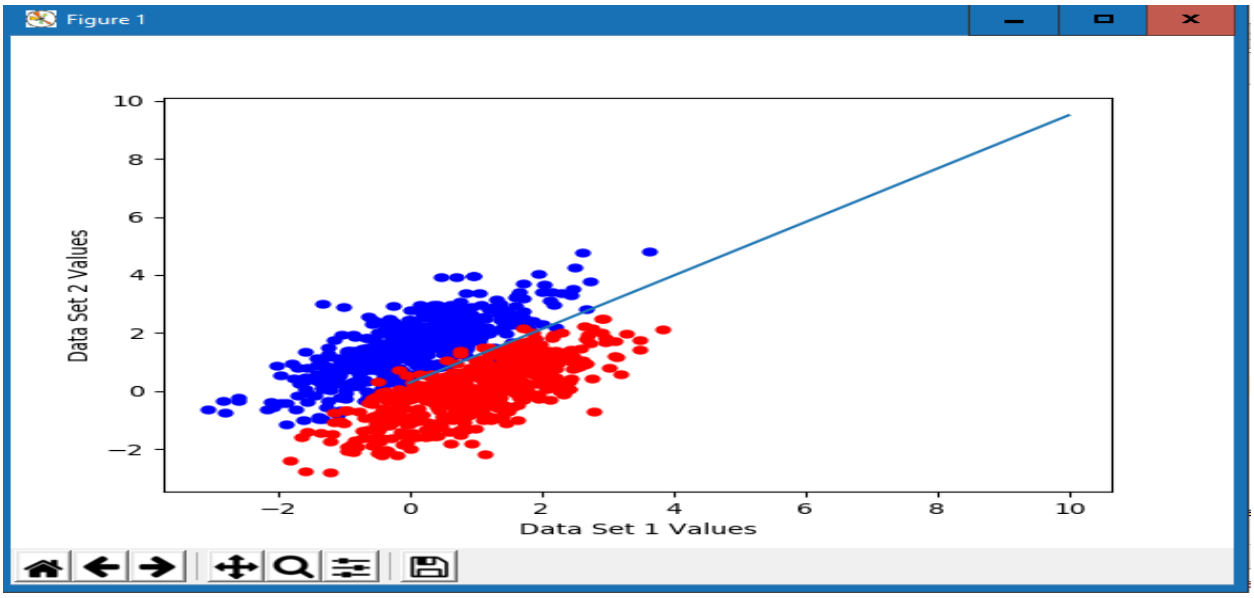
This can be checked by running and printing out the values of the variables NUMB_OF_ITERATIONS val, and wArray in the program. Clearly, we can see very few misclassified data values (points that cross the line boundary or decision boundary).

With maximum number of iterations to 10000 and η = 0.1 and η = 0.01, the nature of the plot stays more or less the same with misclassified data values. The program went through 999 and 100 number of iteration and gave an accuracy of 0.7983 (79%, almost 80%) and 0.8773 (87%) respectively with edge

weights learned: [-857.57737804, -1886.04033945, 2225.38684919] and [-679.63002345, 5052.66244074, 4559.01522697] respectively.

2. Perform online training using gradient descent. Set your learning rate to be η = {1,0.1,0.01}. Set your maximum number of iterations to 10000. How many iterations did you go through your training dataset? What is the accuracy that you have? What are the edge weights that were learned? Compare the learned parameters and accuracy to the ones that you got from batch training. Are they the same? Explain in your report.

Setting the maximum number of iterations to 10000 and η = 1 we have the following



The program went through 999 number of iteration and gave an accuracy of 0.8523 (85%) with edge weights learned: [184.78877003, -7.61381587, 731.68778129]

This can be checked by running and printing out the values of the variables NUMB_OF_ITERATIONS val, and wArray in the program. Clearly, we can see very few misclassified data values (points that cross the line boundary or decision boundary)

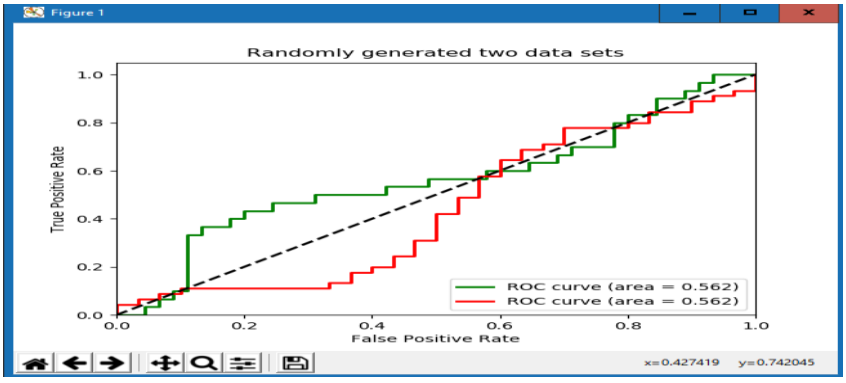
With maximum number of iterations to 10000 and η = 0.1 and η = 0.01, the nature of the graph stays more or less the same with misclassified data values. The program went through 790 and 400 number of iteration and gave an accuracy of 0.9053 (90%) and 0.9536 (95%) respectively with edge weights learned: [-575.99499316, -2237.50907828, 2542.31520778] and [-712.99154625, -2043.42324967, 1533.39009123] respectively.

Comparing the learned parameters to the ones that we got from batch training, we can see that they are different. This is caused by the way the two algorithms calculate weights. The batch training keeps the system weight constant while computing the error associated with each randomly selected data set. On the other hand, with the online training the weights are constantly updated and so the gradient estimation uses different weights for each data sample.

Comparing the accuracy to the ones that we got from batch training (see above), the accuracy for online training is higher than the batch training. Again, according to the lecture by Professor Won Kim this is not always a guarantee and hence makes choosing learning model difficult.

3. Write your own code to draw ROC curve and computing area under the curve (AUC). Evaluate the performance of your LR classifier with batch training with η= {1,0.1,0.01}.

Code: Directory Assignment2 -> q3 -> Roc_Curve.py



The ROC curve plots the true positive rate vs the false-rate as a threshold of the confidence of an instance being positive varied. The broken blue line is what we would expect as the curve for random guessing. However, in this case it is divided the two curves with mirror reflections of each other.

The performance of the logistic regression classifier with batch training differs on the value of η that we use when the values 0.1 then 0.01 the area under the curve (which measure how well a data distinguished between the two data sets) increases/decreases with a wider distribution and fluctuation among the data sets and a poor accuracy performance (0.6 - 0.7). On the other hand, with a $\eta = 1$ we see a fair accuracy performance (.70 - .80). This can be checked by printing out the accuracy for each η of the training data set.

Furthermore, careful analysis shows that the curve is not affected by any changes in class distribution. That is, the curve does not change if the proportion of positive and negative instances in the test set are varied. Ideally, the curve should identify classification thresholds for the data sets with different misclassification values.